Cryptography Acceleration in a RISC-V GPGPU

Austin Adams, Pulkit Gupta, Blaise Tine, Hyesoon Kim
CARRV 2021
June 17th, 2021

Georgia Tech
Motivation

- Our goal: hardware-accelerate SHA-256 and AES-256 in a GPGPU
- Why? Highly parallel cryptography can be useful
  - For SHA, useful for finding collisions / cryptanalysis
  - For AES, parallelize decryption of huge amounts of data, e.g. full-disk encryption
  - Example hypothetical combined use case: file server that encrypts responses with AES and uses SHA for integrity checks
Background and Related Work
Secure Hash Algorithm 2 (SHA-2)

- Family of algorithms that produce a digest (hash) for an input message
- But we focus on SHA-256:
  - Secure hashing algorithm that takes a message and produces a 256-bit digest (or hash)
- SHA-256 uses the following sigma functions \( \sim 64 \) times apiece for every 512-bit message block (ROTR is a right bit rotation, SHR is a right bitshift) [12]:

\[
\begin{align*}
\Sigma_0(x) &= \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x) \\
\Sigma_1(x) &= \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x) \\
\sigma_0(x) &= \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x) \\
\sigma_1(x) &= \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)
\end{align*}
\]
Advanced Encryption Standard (AES)

• AES is a symmetric block cipher
• The input, output, and current state in AES are 16 bytes organized as a $4 \times 4$ column major matrix:

\[
\begin{array}{cccc}
  b_0 & b_4 & b_8 & b_{12} \\
  b_1 & b_5 & b_9 & b_{13} \\
  b_2 & b_6 & b_{10} & b_{14} \\
  b_3 & b_7 & b_{11} & b_{15} \\
\end{array}
\]

• Multiple supported key sizes, but we focus on a 256-bit key, known as AES-256
  • In AES-256, 14 rounds of operations on the 16-byte state
  • “Key expansion” uses the provided key to generate a key schedule with a different key for each column for each round
    • Reused across cipher invocations for the same key [6]
AES-256 Cipher

The core of the AES-256 cipher looks like this [6]:

```python
for round = 1 to 14:
    SubBytes(state)
    ShiftRows(state)
    if round < 14:
        MixColumns(state)
    AddRoundKey(state, keysched[round])
end for
```

- **SubBytes**: Replace each byte according to the S-Box, a predefined substitution table
- **ShiftRows**: Left-rotate the bytes in each row of the state, increasing the offset as we go down
- **MixColumns**: “Mix” together entries in a column by performing shifts and XORs
- **AddRoundKey**: XOR each column with key in the key schedule corresponding to the round and column
Daemen and Rijmen showed you can compute a round of AES (except \textit{AddRoundKey}) using lookups into four tables, each 4 KiB [5]. For AES-256:

$$\begin{bmatrix}
    b_{0,j} \\
    b_{1,j} \\
    b_{2,j} \\
    b_{3,j}
\end{bmatrix} = T_0[a_{0,j}] + T_1[a_{1,j+1 \text{ mod } 4}] + T_2[a_{2,j+2 \text{ mod } 4}] + T_3[a_{3,j+3 \text{ mod } 4}]$$

for each column $0 \leq j < 4$, where $a_{i,j}$ and $b_{i,j}$ are the bytes in the old and almost-new state respectively (still need to perform \textit{AddRoundKey}). Moreover, rotating the result from $T_0$ yields the result of an effective lookup to $T_1, T_2, T_3$. 
AES Cipher Modes

• We implement 3 popular block cipher modes for AES-256 [14]:
  • Electronic Code Book (ECB) - trivial, easiest to parallelize, vulnerable to pattern/replay attacks
  • Cipher Block Chaining (CBC) - less vulnerable, but encryption is serialized (not decryption)
  • Counter (CTR) - easier to manipulate plaintext than CBC, but also easier to parallelize
• GPUs for crypto:
  • First attempt: Cook et al. used the graphics pipeline on a classic GPU to accelerate the S-Box and XORs in AES, but could not beat CPU performance [4]
  • After CUDA release, Manavski wrote a CUDA kernel that beat AES performance on a CPU by $20 \times$ [10]
  • Today, a major use case of SHA-256 (and other hash functions) on GPUs is mining cryptocurrency [1, 9]

• Crypto accelerating RISC-V:
  • Saarinen proposed an RISC-V ISA extension for AES that effectively computes T-table entries at runtime [13], which Marshall et al. recommended for 32-bit RISC-V over other proposals [11]
  • The draft RISC-V cryptography extension now specifies instructions for gathering entropy, SM3, SM4, SHA-2, AES, and some bitwise instructions from “Bitmanip,” another draft specification [15, 2].
    • We implemented a subset of this specification: instructions for SHA-256 and AES-256, plus a bit rotation instruction
Relevant instructions in draft RISC-V cryptography specification:

- **SHA-256**:
  
  - `sha256sum0 rd, rs1`: Performs $rd \leftarrow \Sigma_0(rs1)$
  
  - `sha256sum1 rd, rs1`: Performs $rd \leftarrow \Sigma_1(rs1)$
  
  - `sha256sig0 rd, rs1`: Performs $rd \leftarrow \sigma_0(rs1)$
  
  - `sha256sig1 rd, rs1`: Performs $rd \leftarrow \sigma_1(rs1)$

- **AES-256**:
  
  - `aes32esi rt, rs2, bs`: Uses S-Box on byte $bs$ of $r2$ and XORs result into $rt$. Running 16 times will effect *SubBytes*; choosing registers carefully causes *ShiftRows*; and loading round key into register beforehand achieves *AddRoundKey*
  
  - `aes32esmi rt, rs2, bs`: Performs `aes32esi` plus *MixColumns*
  
  - `aes32dsi rt, rs2, bs`: Inverse `aes32esi`, for decryption
  
  - `aes32dsmi rt, rs2, bs`: Inverse `aes32esmi`, for decryption

- **Bit manipulation**:
  
  - `rori rd, rs1, imm`: Rotate bits in $rs1$ right by immediate and store in $rd`
Vortex is an open-source GPGPU that supports RV32IMF [7].

Microarchitecture:
Implementation
Hardware Implementation

- We add crypto execution unit to handle all new instructions and connect to rest of pipeline
- Use lightweight implementation of S-Box logic proposed by Boyar and Peralta [3]
  - Forward S-Box is 128 gates, 16 deep [3]
  - Based on draft crypto spec reference implementation, but pipelined to avoid stretching cycle time
- Programmed onto an Arria 10 FPGA, generally maintaining original clock frequency
Software Implementation: SHA-256

- We implement three different SHA-256 kernels for Vortex:
  1. “Software”: Pure C implementation based on a naïve reading of the SHA-2 specification [12]
  2. “Hybrid”: Same as software, except with \textit{rori} used for rotations in software $\Sigma_0, \Sigma_1, \sigma_0, \sigma_1$ functions
  3. “Native”: Same as software, except using \textit{sha256sum0}, \textit{sha256sum1}, \textit{sha256sig0}, \textit{sha256sig1} instructions for $\Sigma_0, \Sigma_1, \sigma_0, \sigma_1$ functions
- Each evenly spreads 1 MiB of CPU-generated pseudorandom data across all available threads
Software Implementation: AES-256

- We implement three different Vortex kernels for AES-256 key schedule generation:
  1. “Software”: Pure C implementation based on a naïve reading of the AES specification [6]
  2. “Hybrid”: Same as software, except with \texttt{rori} used for the 7 calls to \texttt{RotWord} made in key schedule generation
  3. “Native”: Same as software, except using \texttt{aes32esi} and \texttt{aes32dsmi} to perform the 13 \texttt{InvMixColumns} calls needed in key schedule generation for the equivalent inverse cipher
  4. “Native+Hybrid”: Both hybrid and native combined

- We implement two different Vortex kernels for the AES-256 cipher:
  1. “Software“: Pure C implementation using a T-tables strategy [5]
  2. “Native”: Same as software except that each cipher round uses the \texttt{aes32esi}, \texttt{aes32esmi}, \texttt{aes32dsi}, and \texttt{aes32dsmi} instructions

- The cipher kernel spreads 2 MiB of CPU-generated pseudorandom data across all available threads (except CBC encryption, which is serialized)
Evaluation

- We ran each kernel on an Arria 10 GX 1150 FPGA against pseudorandom data and collected cycle counts.
- The kernels ran on 16 Vortex cores, each with 4 warps of 4 threads, meaning a total of 256 threads.
SHA-256 Results

• 256 threads hashing 512 total messages, each of size 2 KiB, in parallel. Total 1 MiB
• Hybrid (rori) yields 1.25× speedup
• Native (sigma instructions) yields 1.60× speedup
AES-256 Key Expansion Results

- 1 thread performing key expansion on the 256-bit key
- Native shows the only meaningful speedup, with 3.73× for Native and 4.09× for Native+Hybrid
  - Likely because of the expensive InvMixColumns invocations needed for key schedule generation with the equivalent inverse cipher [6]
AES-256 Cipher Results

- 256 threads hashing 8192 total 16-byte blocks in parallel. Total 2 MiB
- CBC gets $7.02 \times$ speedup for encryption; however, the total runtime is 125.7 to 161.3 times longer than ECB encryption (not shown)
- However, CTR sees $5.19 \times$ and $5.49 \times$ for encryption and decryption, respectively, while maintaining a similar runtime
Physical Characteristics

We largely maintain the clock frequency of the original Vortex, although we see more divergence with larger core counts:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Core(s)</th>
<th>Area Usage (%)</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1</td>
<td>12.86</td>
<td>220</td>
</tr>
<tr>
<td>+ Crypto Unit</td>
<td>1</td>
<td>13.12</td>
<td>218</td>
</tr>
<tr>
<td>Baseline</td>
<td>4</td>
<td>26.48</td>
<td>213</td>
</tr>
<tr>
<td>+ Crypto Unit</td>
<td>4</td>
<td>27.82</td>
<td>208</td>
</tr>
<tr>
<td>Baseline</td>
<td>16</td>
<td>80.24</td>
<td>192</td>
</tr>
<tr>
<td>+ Crypto Unit</td>
<td>16</td>
<td>85.78</td>
<td>177</td>
</tr>
</tbody>
</table>

We believe this may be due to synthesis or place and route issues that future work can resolve.
Implementation Recommendations

• Implementing the entire crypto draft specification may be too expensive on a GPU; however, selectively implementing instructions can yield great results.
• Even with an AES-accelerated Vortex with only 4 cores, we see a $1.42 \times$ speedup over the original Vortex running the software implementation on 16 cores.

Estimated Speedup for AES-256 CTR Encrypt over 16 Original Cores versus Number of Crypto-Accelerated Cores
Future work should:

- Compare performance with other GPGPUs and CPUs, with and without native instructions
- Consider more advanced software implementations, which may reduce speedup
- Determine if our work is vulnerable to timing attacks
- Analyze the impact on our design on the 15nm Vortex chip described in the Vortex paper [7]
Thank you!

Highlight on Cryptocurrencies Mining with CPUs and GPUs and their Benefits Based on their Characteristics.
In *2020 IEEE 10th International Conference on System Engineering and Technology (ICSET)*, pages 67–72, Nov. 2020. ISSN: 2470-640X.


RISC-V bitmanip extension.

J. Boyar and R. Peralta.  
**A Small Depth-16 Circuit for the AES S-Box.**  
In D. Gritzalis, S. Furnell, and M. Theoharidou, editors,  

D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck.  
**CryptoGraphics: Secret Key Cryptography Using Graphics Cards.**  
J. Daemen and V. Rijmen.  
*The Design of Rijndael.*  

**Advanced Encryption Standard (AES).**  
Nov. 2001.  
Last Modified: 2021-03-01T01:03-05:00.

**Vortex: OpenCL Compatible RISC-V GPGPU.**  
L. Ewing.  
**Effects of encryption modes on the tux image.**  
Wikipedia Commons.  
lewing@isc.tamu.edu, produced using GIMP.

**Performance of Hash Algorithms on GPUs for Use in Blockchain.**  

S. A. Manavski.  
**CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography.**  


B. Schneier.  
*Applied Cryptography: Protocols, Algorithms and Source Code in C.*  

*RISC-V cryptographic extension proposals volume i: Scalar & entropy source instructions.*  